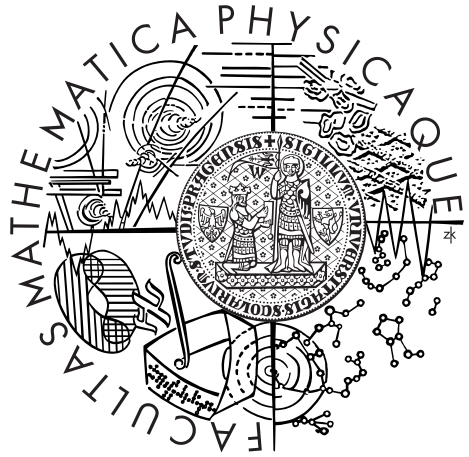
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Tomáš Grošup

Podobnostní vyhledávání obrázků na webu

Department of Software Engineering

Supervisor of the bachelor thesis: RNDr. Jakub Lokoč, Ph.D.
Study programme: Computer science
Specialization: General computer science

Prague 2012

I would like to thank my supervisor, Jakub Lokoč, for the time he has spent with me and for the numerous ideas and advices he gave me. I would also like to thank Tomáš Skopal for a lot of feature ideas, Tomáš Bartoš for supervising the start of this work and David Hoksza for hosting the web application.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date Tomáš Grošup

Název práce: Podobnostní vyhledávání obrázků na webu
Autor: Tomáš Grošup
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Jakub Lokoč, Ph.D.
E-mail vedoucího: lokoc@ksi.mff.cuni.cz

Abstrakt: Předmětem této práce je návrh a vytvoření webového portálu, který umožňuje indexovat a vyhledávat obrázky z dostupných veřejných databází, například z výsledků textového vyhledávače. Portál využívá rychlou extrakci vlastností z obrázků a signaturovou kvadratickou formu pro modelování podobnosti mezi obrázky. Při vyhledávání lze specifikovat různá nastavení a následně srovnat jejich výsledky. Netradiční možností prezentace výsledků je model založený na částicové fyzice, který podporuje exploraci a multidotazy.

Klíčová slova: Podobnostní vyhledávání, signatury, explorace, multidotazy

Title: Similarity based image search on the web
Author: Tomáš Grošup
Department: Department of Software Engineering
Supervisor: RNDr. Jakub Lokoč, Ph.D.
Supervisor's e-mail address: lokoc@ksi.mff.cuni.cz

Abstract: The subject of this bachelor thesis is to design and create a web portal, enabling efficient indexing and content-based searching of images obtained from various free image databases (e.g., results from a keyword-based search engine). The portal provides fast feature extraction technique and for the visual similarity, the signature quadratic form distance is utilized. The search supports various user settings and comparison of their results. Search results can also be presented using a layout based on particle physics, which supports exploration and multi-query.

Keywords: Similarity search, signatures, exploration, multi-query

Contents

Preface

This work focuses on the content-based image retrieval area that actually attracts many research attention all over the world. Although creation of the Smart Image Retrieval web portal was the main goal of this thesis, it has other contributions as well. Some of them were already published on the international conferences [20, 21].

Contributions

Synergistic modeling

To implement a content-based image retrieval system, first a suitable similarity model has to be selected. However human perception of similarity is very complicated and thus the attempts to model it often result in algorithms with a lot of parameters. Given an image database, the suitable similarity model should provide results corresponding to some provided ground truth (e.g., annotated collection). Another key property of the retrieval system is its efficiency and so we focus on the *synergistic modeling*, that can give us either effective or efficient models, or any reasonable compromise depending on all the application's needs.

SIR portal

Smart Image Retrieval, or shortly sir, is a web portal available at siret.cz/sir. The SIR fulfills the implementation goals of this thesis — it represents an efficient content-based search system, and also it lets the user to compare different settings produced by synergistic modeling. As a separate module, a meta-search engine with content-based re-ranking is implemented. It supports both a typical visual presentation of the search result as an ordered list and a novel layout approach using particle physics. The layout supports various exploration capabilities, such as zooming in/out, panning, multi-query refinement, etc. [20, 21].

Structure of this thesis

This thesis is structured into four main chapters. Chapter 1 introduces the reader to image search problematic. It provides definitions of terms used in other chapters and explains

some techniques used in this work. Chapter 2 describes the process of synergistic modeling. It begins with explanation of two different image feature extraction techniques we have modeled, continues with the optimization of various parameters of the feature extractions and ends with presentation of our results. In Chapter 3 we show all important modules of SIR and their functionality on the user level. Chapter 4 is a software documentation showing implementation details such as employed algorithms, data structures and software architecture.

Chapter 1

Image Search problematic

Modern popular image search engines (Google, Bing) utilize text-based search algorithms that not only need each image to be annotated, but the annotation also strongly depends on the person or algorithm that is used to create the annotation. Unfortunately, getting a good annotation for large image collections is still a big problem. Therefore the systems utilize also other orthogonal techniques, for example, the content based image retrieval (CBIR). This approach represents the visual content of the images by descriptors in a feature space $\mathbb{F}$. The image database is then represented as a subset $\mathbb{S}$ of the feature space $\mathbb{F}$.

1.1 Similarity Search

Unlike text or numbers, image descriptors lack natural ordering and cannot utilize the same concepts of the retrieval. Hence, most of the CBIR systems use a search paradigm that employs a total distances function $\delta(o, q)$ defined for two image descriptors $o, q \in \mathbb{F}$. When solving also retrieval efficiency (using indexes), the effective distance function is often not enough. For efficient indexing, the distance function has to satisfy some additional axioms. A popular technique for efficient similarity searching is the metric space approach, where the distance function satisfies metric axioms containing the triangle inequality, that is crucial for metric indexing. Following definitions in this chapter are taken from[1, 20, 22] where you can find also some more details.

Definition 1 *Let $\mathbb{F}$ be a feature space domain, δ a distance function measured on $\mathbb{F}$. $\mathcal{M} = (\mathbb{F}, \delta)$ is called metric space, if distance function δ: $\mathbb{F} \times \mathbb{F} \mapsto \mathbb{R}$ fulfills following postulates:*

(p1) $\forall x, y \in \mathbb{F},\ \delta(x, y) \geq 0$ — *non-negativity*

(p2) $\forall x, y \in \mathbb{F},\ \delta(x, y) = \delta(y, x)$ — *symmetry*

(p3) $\forall x \in \mathbb{F},\ \delta(x, x) = 0$ — *reflexivity*

(p4) $\forall x, y \in \mathbb{F},\ x \neq y \Rightarrow \delta(x, y) > 0$ *positiveness*

(p5) $\forall x, y, z \in \mathbb{F},\ \delta(x, z) \leq \delta(x, y) + \delta(y, z)$ *triangle inequality*

Having such an abstraction for the similarity model, we can investigate techniques that will work in a large number of applications across several research areas.

1.2 Distance functions

Distance functions are used to quantify the closeness of objects and thus measure their dissimilarity. It is up to the domain experts to choose (and eventually parameterize) a suitable distance function that should mostly correspond to user expectations. In this section, we will show two different distances used in this thesis.

Minkowsi metrics

The Minkowski metrics (or L_p metrics) are the most popular dissimilarity measures used in various applications. However, the metrics are restricted just to vector spaces, where a distance between two vectors (points) is computed.

Definition 2 *Let $\mathbb{V}$ be an n-dimensional vector space and $x, y \in \mathbb{V}$, then an L_p metric is defined as:*

$$\mathrm{L}_p(x, y) = (\sum_{i=1}^{n} |x_i y_i|^p)^{\frac{1}{p}}$$

Only for $p \geq 1$ holds that L_p is a metric, for $p < 1$ it does not satisfy the triangle inequality. L_1 distance is known as the Manhattan distance, the L_2 metric is the Euclidean distance and the L_∞ is called the Chessboard distance.The time complexity of the distance evaluation is $O(n)$, hence L_p metrics are considered as cheap dissimilarity measures. The L_p metrics are suitable to model a dissimilarity in vector spaces with independent dimensions.

There also exist cases, where it is profitable to prefer more significant coordinates and to suppress the less significant ones. For such cases, the weighted L_p metric can be used as a generalized variant of the original L_p metric[22].

Signature Quadratic Form Distance

The Signature Quadratic Form Distance is a generalization of the Quadratic Form Distance[22] for feature signatures. It is defined as follows:

Definition 3 *Given two feature signatures $S^q = \{\langle r_i^q, w_i^q \rangle\}_{i=1}^n$ and $S^o = \{\langle r_i^o, w_i^o \rangle\}_{i=1}^m$ and a similarity function $f_s : \mathbb{F} \times \mathbb{F} \to \mathbb{R}$ over a feature space $\mathbb{F}$, the signature quadratic form distance SQFD_{f_s} between S^q and S^o is defined as:*

$$\mathrm{SQFD}_{f_s}(S^q, S^o) = \sqrt{(w_q \mid -w_o) \cdot A_{f_s} \cdot (w_q \mid -w_o)^T},$$

where $A_{f_s} \in \mathbb{R}^{(n+m)\times(n+m)}$ *is the similarity matrix arising from applying the similarity function* f_s *to the corresponding feature representatives, i.e.,* $a_{ij} = f_s(r_i, r_j)$*. Furthermore,* $w_q = (w_1^q, \ldots, w_n^q)$ *and* $w_o = (w_1^o, \ldots, w_m^o)$ *form weight vectors, and* $(w_q \mid -w_o) = (w_1^q, \ldots, w_n^q, -w_1^o, \ldots, -w_m^o)$ *denotes the concatenation of weights* w_q *and* $-w_o$*.*

The similarity function f_s is used to determine similarity values between all pairs of representatives from the feature signatures. In our implementation we use the similarity function $f_s(r_i, r_j) = e^{-\alpha L_2(r_i, r_j)^2}$, where α is a constant for controlling the precision-indexability tradeoff, and L_2 denotes the Euclidean distance.

The time complexity of SQFD is $O(n^2 * T_{f_s})$, where T_{f_s} is the time complexity of the f_s similarity function. The time complexity of the SQFD can be a performance bottleneck of the retrieval, especially when using larger feature signatures. Therefore the number of SQFD computations becomes an important performance measure in the metric indexing techniques (even the traditional I/O costs become negligible).

1.3 Popular queries

Besides many others, there are two most popular similarity queries often used in similarity search tasks — the range query and the nearest neighbor query (KNN).

Definition 4 *Let* q *be a query object in domain* $\mathbb{F}$ *and* r *be a distance. Range query is defined as* $R(\mathrm{q}, \mathrm{r}) = \{o \in X, \delta(o, q) \leq \mathrm{r}\}$*, where* δ *is a metric function on domain* $\mathbb{F}$ *and* $X \subseteq \mathbb{F}$*.*

Definition 5 *Let* q *be a query object in domain* $\mathbb{F}$ *and* k *be a number of wanted results. The* k *nearest neighbor query is defined as* $\mathrm{k}NN(\mathrm{q}) = \{R \subseteq X, |R| = \mathrm{k} \wedge \forall x \in R, y \in X - R : \delta(q, x) \leq \delta(q, y)\}$*, where* δ *is a metric function on domain* $\mathbb{F}$ *and* $X \subseteq \mathbb{F}$*.*

As we can see in Figure 1.3, both similarity queries are represented by ball region, where the radius of the ball is dynamically adjusted for the KNN query.

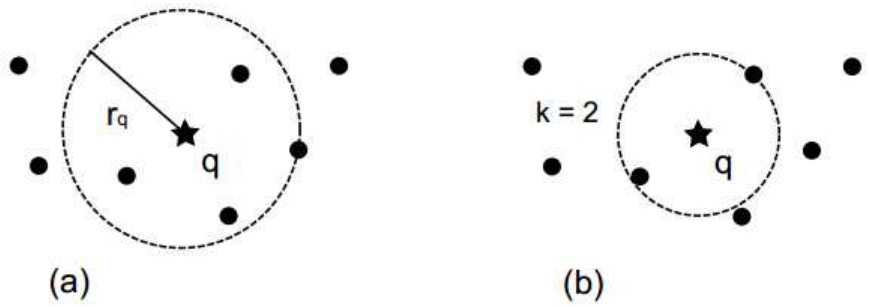


Figure 1.1: : (a) Range query ball with radius r_q (b) 2NN query

1.4 Indexability

The triangle inequality axiom provides us an option to correctly lower-bound an unknown distance by use of the distances that are already known[1, s. 34]. This allows us to build an indexing structure reducing costly distance computations for a query to a minimum. Especially in the case when the expensive SQFD distance is employed, because the reduction of the number of distance computations can significantly speedup the search, while the additional overhead of the index structure is negligible.

However, the number of distance computations, that can be omitted, depends also on the distribution of distances in the distance space. The so called indexability can be determined from the distance distribution histogram by an indexability indicator, e.g., by the *intrinsic dimensionality[19], [15]* value(iDim). It is defined as $\rho = \frac{\mu^2}{2\sigma^2}$, where μ and σ^2 are the mean and the variance of the dataset's distance distribution histogram[23].The lower this value is, the better the index structure works. Figure 1.2 shows two examples of distance distributions. The second distribution cannot utilize the triangle inequality to reduce distance computations at all.

An efficient yet simple index [16] is the *pivot table* (its original name is *Linear AESA*). It works in main memory and uses m pivots to lower-bound distance computations. Distances from all n database objects to pivots are stored in a $m \times n$ matrix and the search complexity of a single query is $m + O(1)$ distance computations[1].

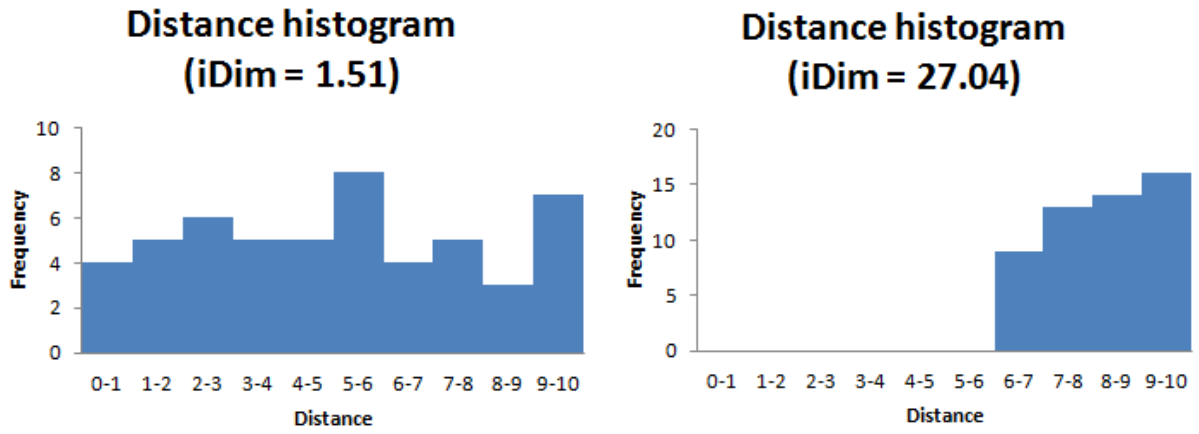


Figure 1.2: Examples of distance distributions and their iDim

Chapter 2

Synergistic Modeling

First task to solve when implementing content-based search functionality is to select a suitable similarity model. We have utilized the feature signatures and the Signature Quadratic Form Distance to fulfill one of the goals of this thesis. We have utilized one feature space and implemented two different algorithms to create feature signatures from images. All the employed algorithms utilize a lot of parameters during the feature extraction and during the distance evaluation. We have inspected the parameter space in order to create a similarity space exhibiting high retrieval precision, good indexability and a reasonable computation time of one distance. Having no mathematical dependencies between parameter values and similarity space precision, finding the optimal parameter values was a black-box optimization problem. Since our requirements are known to be contradictory[1], we had to choose several results with the most-suitable combination of precision, indexability and speed.

2.1 Feature Extraction

Our feature extraction focuses on three primitive features — color, position and texture. For color representation we use the *CIE*[2] *LAB* color space, which was designed to approximate human vision[3].

2.1.1 Feature signature

The feature signatures are popular tool for flexible image description, where the main advantage of using feature signatures is their variable size, which makes them suitable for both simple and complex images.

Definition 6 *Given a feature space* $\mathbb{F}$*, the feature signature* S^o *of a multimedia object* o *is defined as a set of tuples from* $\mathbb{F} \times \mathbb{R}^+$ *consisting of representatives* $r^o \in \mathbb{F}$ *and weights* $w^o \in \mathbb{R}^+$

[1] High precision results in slower computation time

[2] *International Commission on Illumination (Commission internationale de l'éclairage in French)*

Each representative r^o from a feature signature represents an area with a central point. The feature space we use in this work for feature signatures is a 7-dimensional vector of real numbers obtaining the following components:

X. The x-coordinate

Y. The y-coordinate

L. Lightness, L* in *CIE LAB* color space

A. Color position between magenta and green, a* in *CIE LAB* color space

B. Color position between yellow and blue, b* in *CIE LAB* color space

C. Contrast

E. Entropy

All values are normalized to $\langle 0, 1 \rangle$. To adjust importance of individual dimensions of our feature we used weighted L_2 metric for our 7-dimensional features as the ground distance between two representatives.

The feature signatures can be compared by SQFD function which needs $O(n^2)$ computations of the ground distance. For fast online processing, signatures having more then 100 features were too big to be used with the SQFD function. Thus keeping the signature small was the primary goal when creating an extraction algorithm.

2.1.2 Clustering-based extraction

First idea of an extraction algorithm was to extract a few important clusters from each image, as we can see in Figure 2.2. In Figure 2.1.2 we can see four main parts of this algorithm. The algorithm begins with preprocessing the image and extracting our 7-dimensional feature from a few thousands pixels. Those pixels are generated using the Gaussian random number generator with its mean in the middle of the image. That way we simulate focusing the mid area of the image and paying only a small attention to the edges. After generating the features the algorithm starts to reduce their amount with several iterations of clustering. In each iteration, features close to each other in weighted L^2metric space are merged into one with a summed weight. In addition, each iteration level has a threshold mechanism which discards all features having weight less than a specified minimum. This minimal value grows with each iteration in order to eliminate noisy pixels/areas from the original image.

After each iteration, the set of features is a valid feature signature. With each iteration, the size of the signature is lower and lower. This gives us not only the possibility to stop at a specified size of the signature, but also to generate more feature signatures during one run of the algorithm. This was handy when finding the iteration count resulting in the best similarity search precision.

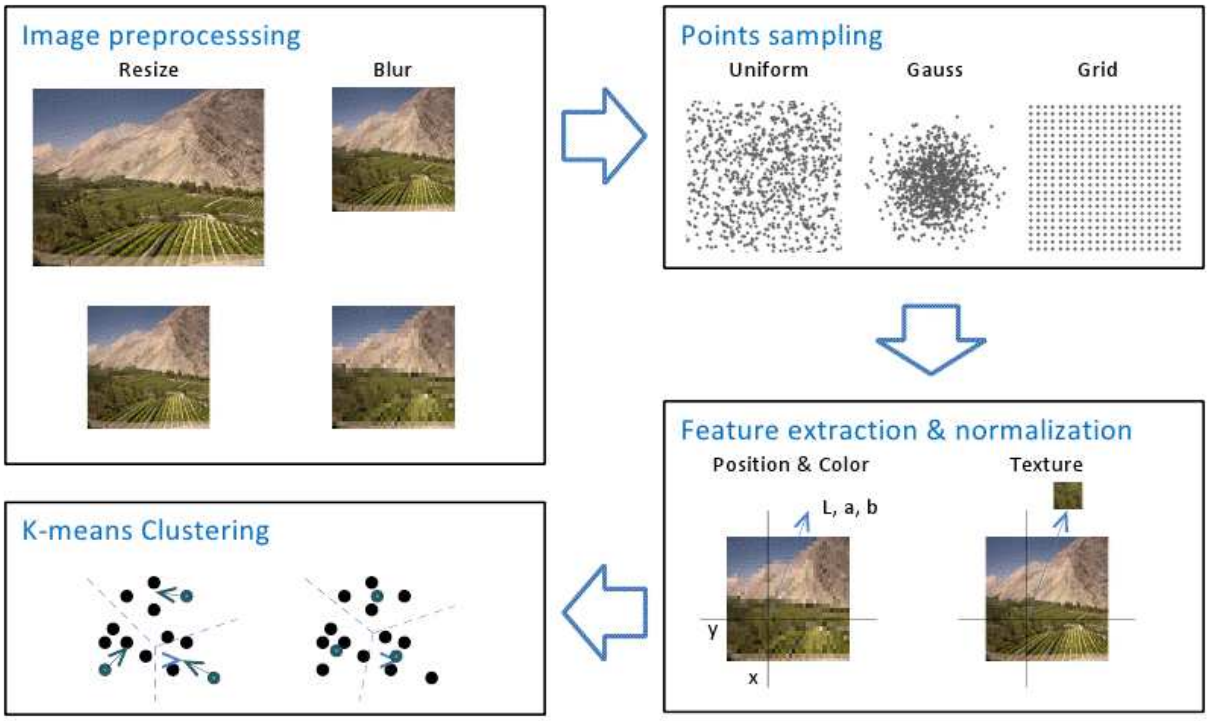


Figure 2.1: Clustering extraction algorithm

The algorithm is quite complex and spends a lot of time in early iterations when a lot of features are still present. But thanks to the clustering, extracted signature is both compact and results in a very good retrieval precision. It has a varying number of centroids and the signature fits the image, as we can see in Figure 2.2.In the scenarios where search happens significantly more often then extraction, the extraction time is not a big problem. Also the algorithm speed can be further enhanced by use of the parallelization on a multi-core CPU or even on a many-core GPU architecture[2].

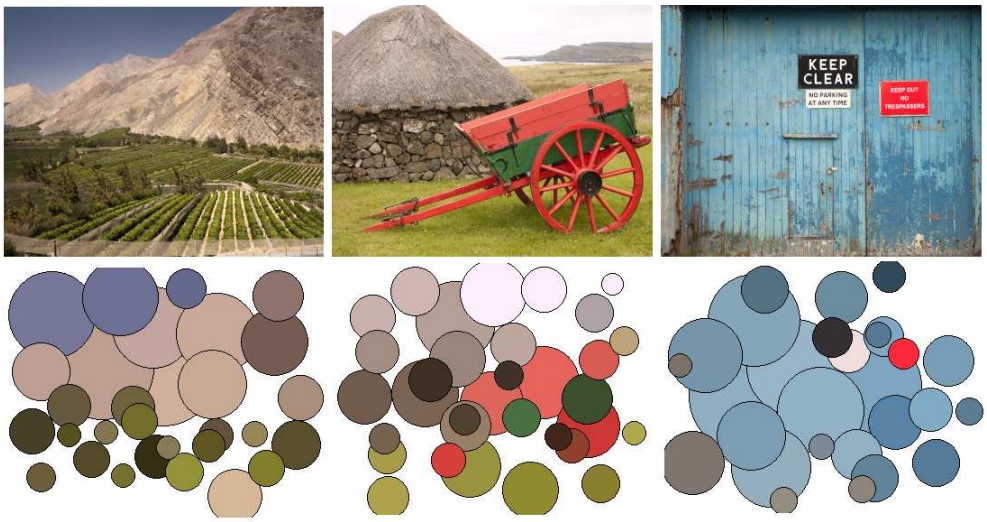


Figure 2.2: Feature Signature

2.1.3 Non-clustering extraction

To implement the meta-search functionality in the *SIR* engine we needed a different extraction algorithm. The algorithm in previous section was designed for a typical database use where the extraction happens once during a preprocessing phase and the feature signatures are stored in a persistent storage. Therefore it is not a big problem that the algorithm consumes a lot of CPU time. In meta-search engines, on the other hand, we have to run extraction for each image with each unique request, because we do not have any database with already extracted signatures. For this scenario we need an algorithm that can extract signatures from one thousand images in a very short time, because the user is willing to wait for the results only for a few seconds (and here any preprocessing is impossible). We created a simple but fast extraction algorithm which could be improved to provide a better retrieval precision.

The initial idea was to create an extremely small thumbnail (10x10) and convert every pixel of it to a single feature. As a side effect, conversion to a thumbnail also serves as noise reduction technique. To save even more time, we have also omitted computing contrast and entropy. However this led to similarity models strongly depending on image background and so we needed a way to prioritize the mid area of the image. We came up with two independent techniques which are now used together:

1. Assigning a weight between zero and one to each pixel. We decided to employ the Gaussian distribution with mean it the middle of the image. The Gaussian peek's gradient is controlled with the variance parameter and can even result in weights zero at the edges, which means that the edges are not considered at all.

2. Creating two separate thumbnails of different sizes and combining them together. The one with the smaller resolution is used only for the pixels at the edges and we

call it the *outer thumbnail*. The other one is used for the rest of the image and it's called the *inner thumbnail*. This gives us not only the opportunity to focus on the mid of the image by using more pixels from the center, but it also reduces the signature size when using even smaller resolution for the outer thumbnail (e.g. 6x6).

This extraction produces quickly a fixed amount of centroids and their positions are also fixed. However, information about the texture and smaller objects is lost, as we can see in Figure 2.3.

Figure 2.3: Non-clustering extraction

2.2 Optimization

To reach a better similarity search precision we had to inspect the parameter space of our algorithms. To lower the number of the inspected parameters, we optimized some of them individually and we set some to a constant by a design decision. As a result, the following parameters were optimized for both algorithms:

Alpha parameter for SQFD function. This parameter does not occur during the extraction itself so we could first extract signatures for entire data set and then run measurement for several values of alpha.

Vector of weights for our 7-dimensional feature. This vector is used in the ground distance in SQFD and also in the clustering phase in the clustering-based extraction. We reduced the initial dimensionality of seven to four by assigning the same weight to X and Y, A and B and to C and E.

In addition to those common parameters, we optimized the following in the non-clustering extraction algorithm:

Inner thumbnail size.

Outer thumbnail size.

Thickness of the edge used from the outer thumbnail.

Variance parameter for weights assigned to each pixel.

And this one in the clustering-based extraction algorithm:

Iterations of clustering phase, divided into two options: Detailed signature (few iterations) and general signature (more iterations).

Each possible configuration of parameters results in an individual MAP (mean average precision[19]), iDim and time needed for all SQFD computations. We didn't really want a single best configuration, since requirements for iDim or SQFD computation time can vary in different scenarios.We didn't even have a single number summarizing a configuration's quality. In other words, we lacked a fitness function. That is why we could not use a genetic approach. What we did instead was generating a lot of random configurations and then exploring the results. Then given a concrete requirement for indexability we can explore the results and find the one fulfilling the requirement and resulting in the smallest precision loss. This is what we call the Synergistic Modeling.

Assuming the multidimensional space defined by all possible configurations is continuous, we can further enhance MAP of a chosen configuration by what we call the Gradient Method. This method generates neighbors of the configuration by changing a single parameter by a small value ε in both directions, doing this one by one for all parameters. The method repeats itself for the best neighbor with a greater MAP until there is no such neighbor.

2.3 Results

We ran our optimization script for both our extraction methods and for three datasets. In this section we will focus only on a few global observations, the extensive results and concrete configurations can be found in the attachments.

For the clustering-based extraction, we started with the Amsterdam Library of Object Images (ALOI)[24]. It contains 1,000 small objects, each captured under multiple configurations achieved by changing the illumination and viewpoint. Two images are considered similar if they show the same object. This dataset is very good for testing the precision of near-duplicate detection. Our best configuration produced a result with 78.6% MAP and 4.14 iDim, which is a very precise and well-indexable result. Figure 2.4 shows the results as iDim/MAP trade-off chart. As we can see, we can achieve even better iDim result with just a small drop of MAP.

The second dataset we tried for clustering-based extraction was the Profimedia dataset[3] created at Masaryk University in Brno. This dataset provides 100 query objects with a ground-truth for each of them. It was constructed to be a benchmark for content-based image retrieval systems and we used it as our main benchmark for the clustering-based extraction. The best configuration achieved 33.1% MAP and 19.91 iDim. Such result is not indexable, but there are other configurations with a better iDim and a small drop of MAP. The best candidates would be the configurations on the skyline of the iDim/MAP trade-off chart on Figure 2.5. We also investigated the effect of clustering iterations on the resulting precision. Each iteration reduces number of centroids and thus reduces the time needed for one SQFD computation. As the Figure 2.5 shows, there are results with good MAP even for small values of SQFD time, which is the time needed for all distance computations in seconds. This means that we can speed up our distance computation in exchange for more time spent in the clustering phase during signature creation. The last thing we focused on was the effect of the alpha parameter for SQFD. As Figure 2.6 shows, in average higher alpha always means greater iDim. But then we focused only on the result with best MAP for each value of alpha. It turns out, that the effect of alpha isn't clear and we really need to run more experiments to find the best option for each configuration.

For non-clustering extraction, we did not focus on iDim. This extraction was created for online re-ranking and instead of indexability we were interested in time needed for all SQFD computations. We also changed the dataset to TWIC, that is an image collection with ground-truth created using keyword search at Google Images and consecutive human-based filtering. The TWIC dataset suits well as a benchmark for online re-ranking of the results obtained from servers like Google Images.

When using non-clustering extraction, we have to determine the optimal number of centroids. Hence, for each parameter configuration we computed the fixed number of centroids it uses in feature extraction. Then we found best MAP achieved for each such number and corresponding SQFD time, as we can see in Figure 2.7. The best result was found for 276 centroids with 33.9% MAP and SQFD time of $406s$ for all distance computation, however we also found a result for 76 centroids with 30.4% MAP and $31s$ SQFD time, which is 13×faster.

[3] http://mufin.fi.muni.cz/tiki-index.php?page=Profimedia+evaluation+platform

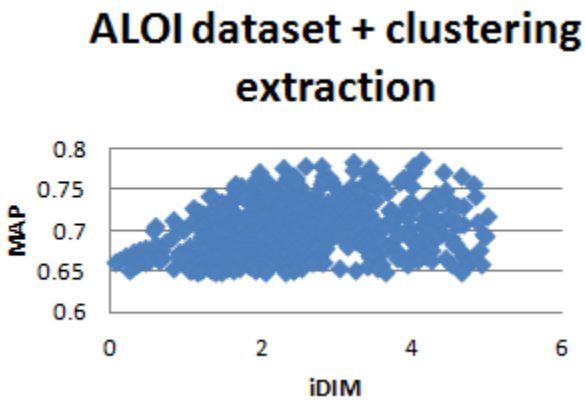


Figure 2.4: ALOI - iDim/MAP trade-off.

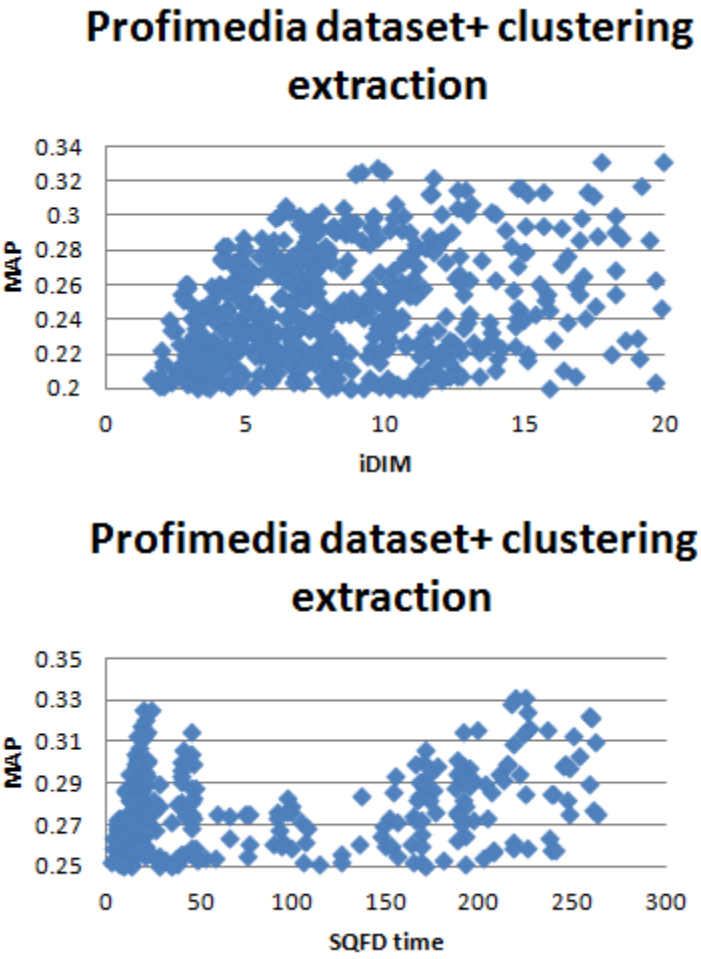


Figure 2.5: Profimedia - trade-off charts.

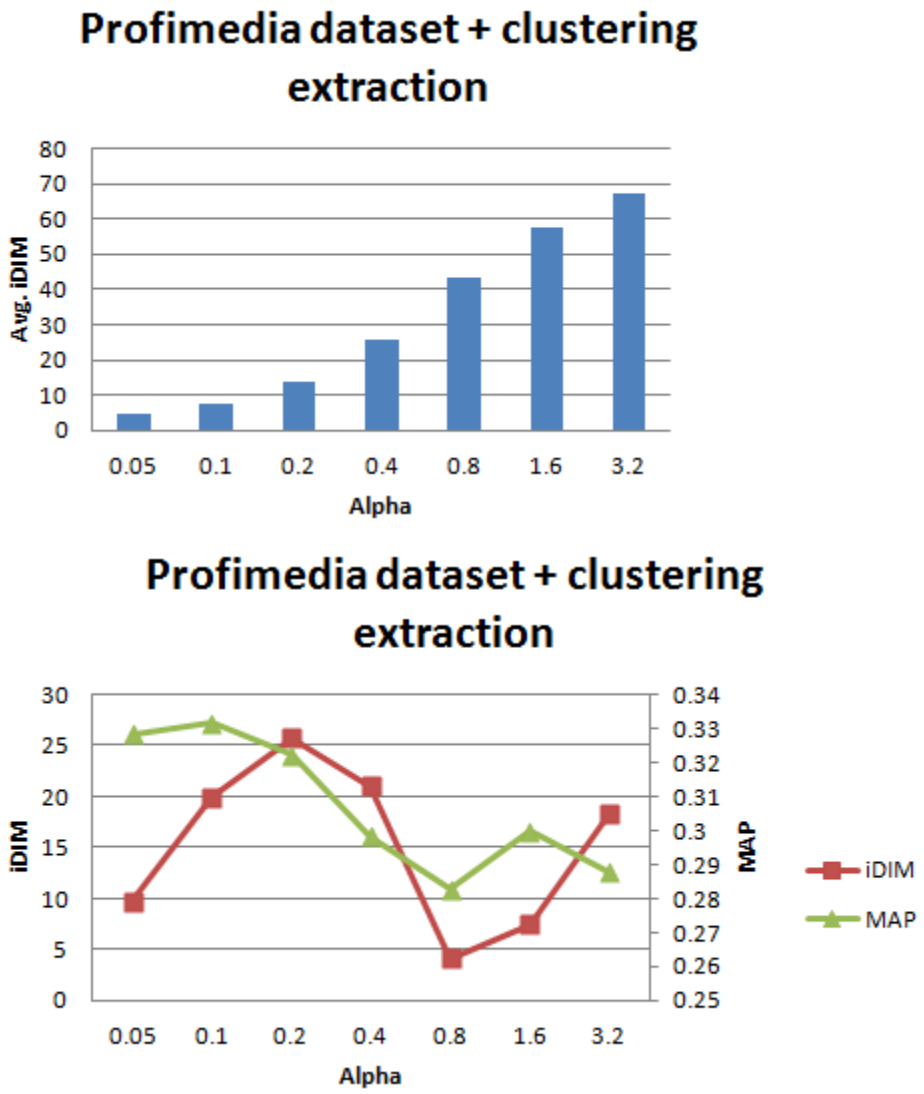


Figure 2.6: Profimedia - alpha parameter

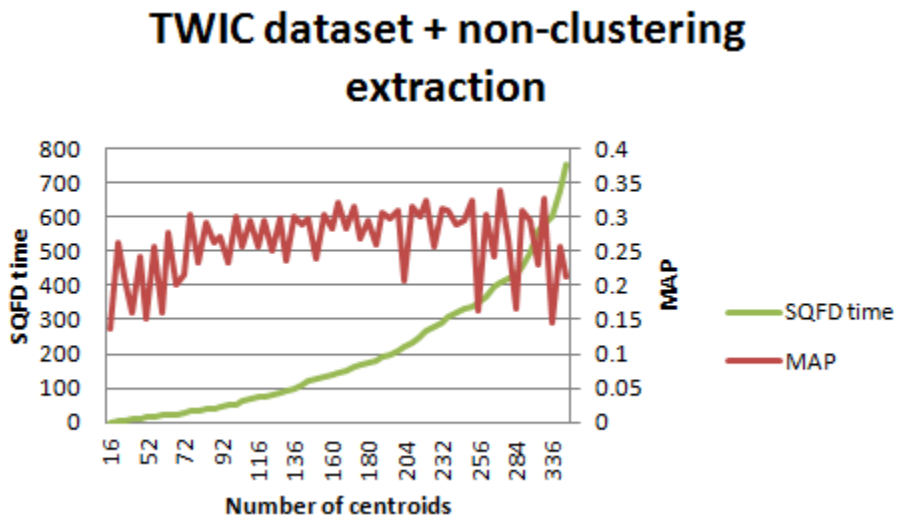


Figure 2.7: TWIC - number of centroids

Chapter 3

Smart Image Retrieval portal

Smart Image Retrieval portal is a demo application showing various aspects of image retrieval. To the best of our knowledge, it is the first demo using feature signatures and SQFD. Its first module serves as an image search framework, where you can easily add a new search method just by implementing a simple interface. Then you can compare different implementations on search precision, query performance or number of database operations. The breakthrough in the area is using particle physics model to show the search result. This search result is obtained via re-ranking of a third-party search engine and enables the user to specify his intention by exploration and the multi-query refinement. That is the second module of the Smart Image Retrieval engine.

Content-based image retrieval engines

There are plenty of content-based image retrieval engines on the web, both commercial and research projects[1]. Most of them provide query-by-example or query-by-sketch query modality. Some search engines also allow filtering of the search result by specifying desired color, size or even type of the image. They usually maintain a large database of images and need to extract and store various information obtained from the image. The typical approach is to extract global descriptors representing color, shapes and texture. Those are usually a subset of image descriptors defined in the MPEG-7 specification [2] or in-house developed descriptors.

To keep their database up-to-date, commercial applications also need to regularly scan all possible image sources and download new data to the database. This is a very challenging task and is often done together with crawling of entire websites. As [4] state, such systems must be highly scalable and require a very complex architecture and expensive hardware.

[1] Visit http://en.wikipedia.org/wiki/List_of_CBIR_engines to see a list of them

[2] http://mpeg.chiariglione.org/standards/mpeg-7/mpeg-7.htm

3.1 Static search

First module of SIR, Static search, is a compact content-based image retrieval framework. It does not store any copies of the images, which can often be a violation of the copyright law, it only stores links to the original images. Its main purpose is academic research of different implementations and their comparison. That is why we do not need to maintain a large and up-to-date database.

Our database is filled with links to original images and their meta-data, such as textual description or the original website where the image was found. Currently the application supports two ways of importing images:

1. Importing Flickr[3] images for a given keyword.
2. Using user-created snapshots. The process of creating them is described in 3.3 on page 29.

Each imported image is processed by all currently installed implementations. That usually means extracting its features, saving them to the database and updating the database's index. This is transparent to SIR, since every installed implementation is only a plug-in communicating with the application via a thin interface.

3.1.1 Query by example

Static search module provides two basic mechanisms to enter the query. The first is using an image shown to the user in the search results. It then displays 10 images most similar to the selected image, including the source image itself. To have some start configuration, the page offers a random selection of images from the database. The second mechanism is the option to upload an own image and select the rectangular area of interest in the image (that can be the whole image). After that the user chooses any combination of currently installed plug-ins, each being a separate similarity search implementation. The application answers with lists of 10 nearest neighbors results, one list for each selected plug-in.

3.1.2 Experimental zone

The *Experimental zone* was created to provide a user friendly environment for testing and comparing different similarity search models. The models are displayed as independent widgets that can be re-ordered, deleted and run for a currently set query image.

In addition to all installed plug-ins, the user can also create his own similarity search model by combining two different plug-ins and by the use of re-ranking. The first selected plug-in performs a KNN query returning K objects. Then the second one uses those K objects to perform a 10NN query. This can not only result in better precision, but it could be a huge performance enhancement. The first plug-in might be a similarity model with

[3] http://www.flickr.com/

very cheap distance function but a lower precision and the second plug-in a very precise but not indexable similarity model based on SQFD. With an appropriate value of K the result still has the quality of the precise model and utilizes the speed of the fast model.

Since our image database also holds meta-data, we created a plug-in called the *Text based cheater*. It returns images having the same textual annotation and can be used in re-ranking to filter out non-related images or separately as a ground-truth for computing precision and recall.

When creating other plug-ins we tried different parameter configurations obtained from synergistic modeling. In addition, we turned one similarity space into a non-metric by breaking the triangle inequality and utilizing an approximate indexing method ([18, 17]). It has been shown that such approximate method can reduce distance computations by two orders of magnitude and change the search precision only by a small amount.

3.2 Online re-ranking

Online re-ranking is the second module of SIR. It consists of several independent components, which can be replaced or reused in a different application.

3.2.1 Meta-search web engine

Instead of maintaining a huge database of images, we have developed an image meta-search web engine. In this engine, the keyword query is passed to an image search engine using full-text search, *Bing Images* [4]. All the full-text search, huge database maintenance and keeping the content up-to-date is thus done by a third-party application. We can even use more image sources at once and combine their results together. As a side effect, we can also utilize advanced query constructs of the engine. For example, we can restricting the search to a specific domain or use Boolean logic.

After retrieving the search results as a ranked list, we re-rank the images based on visual content. This technique is not new and has been investigated before to refine search results with a low quality and to learn image categories ([7, 6, 5]). The authors of [6] used local features of gray-scale images ([8]) to find the most relevant objects and discarded the rest to form their refined search. We, on the other hand, compute similarity between all pairs of images using feature signatures to form an exploration structure, in which every image is accessible and visually similar images are displayed together. The difference is most noticeable if the query word is a homonym [5] or a homograph[6].

To save expensive distance computations between feature signatures we build a small pivot table using an greedy pivot-selection algorithm similar to one proposed in [9]. To select m pivots, we run the algorithm on all n objects instead of randomly selected $3m$ objects as proposed by the author of [9], because we would eventually need to compute the

[4] http://www.bing.com/?scope=images\&FORM=Z9LH

[5] Homonym : Same spelling and pronunciation, but different meaning

[6] Homograph: Same spelling, but different pronunciation and meaning

additional distances anyway. Having a $m*n$ pivot table, we can approximate all expensive SQFD distances by use of the cheap permutations distances. Let us denote, that each permutation is created from distances to m pivots sorted in ascending order.

3.2.2 Dynamic visualization

Our dynamic visualization based on particle physics model is a new presentation layout method for image search results.

Having distances between all pairs of images, we construct an undirected weighted graph. Each vertex represents one image and each edge represents similarity between two images. To filter out some edges, we have a threshold mechanism for discarding edges with a low weight. We either allow only edges with weight above a specified minimal value, or we keep only edges to several nearest neighbors for each vertex.

To present such graph to the user, a drawing algorithm is needed. Finding the optimal drawing for various quality measures such as symmetry or crossing number[7] is computationally expensive or sometimes even NP-hard ([10]), which would require more powerful hardware. To avoid this overhead, we used a force-directed graph drawing algorithm presented in [11] and implemented by [12]. The algorithm uses particle physics model to improve the graph arrangement in each of its iterations by using attractive and repulsive forces. Repulsive forces exist between each pair of objects and attractive forces are based on edges, in our case the similarities between images. The algorithm is iterative and the number of iterations before the graph is stabilized is unknown. To solve this issue, the algorithm runs in the user's browser and continuously shows its state. The first impression is very quick and small motion later involved is not disturbing the user. If the graph gets stuck in a local maximum of layout quality, the user can move problematic object with his mouse and resolve that issue.

As we can see in Figure 3.1, the results are very promising.

3.2.3 Data-set exploration

Full-text search engines return hundreds of interesting images for a typical image search request. It is impossible for the user to process such a large amount of information at once and find the best answer to his/her query. To keep displayed set of images in a reasonable size and still provide an option to quickly access desired result we have utilized the dynamic exploration. In splits the presentation into two different phases:

Initial view. To have something to start with, we choose a random sample of a small size ($<$50) and visualize it as a graph. We call it the *Zoom-out* view. The graph drawing algorithm takes care of clustering the images into separate categories and the user quickly sees what types of images are available for his query.

[7] Number of pairs of edges crossing each other.

Figure 3.1: Results for keyword "jaguar"

Exploration. Once he finds the category he is interested in, he can begin with exploration by selecting an image. This results in a detailed view with the selected image and k of its neighbors. Same as before, this set is displayed as a graph which forms visually similar clusters. This is called the *Zoom-in* view. The user can continue with exploration by selecting any of the displayed images or by returning to the *Zoom-out* view and exploring a different part of the data-set.

The model of our exploration is flat, because we always show k nearest neighbors for each image and don't have any deeper hierarchy. This can be changed in the future by utilizing a tree structure such as M-tree[14] or PM-tree[13].

Figure 3.1 is the initial *Zoom-out* view for the query keyword "jaguar". Figure 3.2.3 shows a *Zoom-in* view after selecting a black car.

Figure 3.2: Exploration

3.2.4 Multi-query

Even with the use of exploration, the resulting view still contains some noisy images. As we can see in Figure 3.2.3, the *Zoom-in* view for a car shows an jaguar animal, too. The presence of the animal in k nearest neighbors of a car can be caused either by the inaccuracy of our similarity model or by a small amount of black cars in the data-set ($< k$). To filter-out those noisy images and allow a better specification of the search intent, we make use of multi-query to extend our exploration functionality. The multi-query technique does not limit the number of query images. To better balance the multi-query intent, we also use a weight parameter for each query image that indicates its importance.

Every time a change occurs in the configuration of the multi-query, we display the result for the current multi-query immediately. This leads to fast user experience and a sequential building of the multi-query set by adding new images or changing their weights until all the noisy images are filtered out.

Figure 3.2.4 shows a result for a multi-query defined by two cars, one white and one black. As we can see, the result contains no more animals and forms visually similar clusters.

3.3 Snapshots

Snapshots component is the third module of the SIR portal, which brings results from Online re-ranking module to Static search module. Using snapshots module, we can simply create an annotated image collection in a user-friendly way and make that collection accessible, so users can share interesting image search results with their friends or view them later. The life-cycle of a snapshot can be divided into 3 parts:

Selecting images. In the Online re-ranking module, every view can be turned into a snapshot. With the help of exploration and multi-query techniques, users can concrete their search intent and filter out noisy images. They can also modify the view by changing the k parameter for the KNN query used in the exploration.

Saving the snapshot. Each snapshot must be saved with a textual annotation. Once it is saved, the user gets persistent link to his snapshot.

Import to Static search. The administrator of the SIR portal can later import snapshots to the Static search module by one simple click. Annotated images are then used to train and compare our image search algorithms.

Figure 3.3: Multi-query

Chapter 4

Implementation

4.1 Basic overview

The main programming language of this project is *C# 4.0* with *.NET framework 4.0*. To browse or run the source code we recommend using *Visual Studio 2011*. SIR portal is an *ASP.NET MVC 3* application. To run it inside Visual Studio you need to install this[1] web framework first. You will also need *Entity Framework*[2] and *Sql Server Compact*[3] or *SQL Server 2008* to get the database working. The database connection can be changed inside the *web.config* file. Default connection uses *Sql Server Compact* database, which *Entity Framework* automatically creates with all necessary tables. In the same config file you will also need to change all paths in the *appSettings* section to match your system. After that, SIR portal can be deployed to an *IIS Web Server* via standard *Visual Studio* publish options.

4.2 Feature signature optimization

Feature signature optimization is a separate console application. It is an one-time script and can be configured only via changes in the source code. It needs a data-set folder and produces a *.csv* file with results. The optimization process can be stopped by placing a *stop.txt* file in a specified folder.

4.2.1 Non-clustering extraction

The only computationally expensive part in the algorithm of non-clustering feature extraction is the assignment of weights based on Gaussian distribution, since it requires evaluation of $\exp(distance^2/variance)$. Those values are calculated just once for all images and then

[1] http://www.asp.net/mvc/mvc3
[2] http://nuget.org/packages/entityframework
[3] *http://msdn.microsoft.com/en-us/data/ff687142*

stored in the *weights* array. To retrieve those weights in the right order it is necessary to use *iterateStructure* method, first for *outer thumbnail* and then for *inner thumbnail.*

4.2.2 SQFD Performance enhancements

Before our enhancements, the SQFD computation was the bottleneck of the Online re-rank module. We improved it in several ways:

- Using unsafe C# for data in arrays. It uses pointers and drops all runtime boundary checks. Unlike objects created with the *new* keyword, unsafe arrays are allocated on the stack and aren't collected with the garbage collector.

- Caching partial computations. The similarity matrix has $(m+n)*(m+n)$ entries where n and m are the number of features in FS_1 and FS_2, the two operands of SQFD. The matrix can be divided into 4 parts representing $FS_1 * FS_1$, $FS_2 * FS_2$, $FS_1 * FS_2$ and $FS_2 * FS_1$. Thanks to the symmetry of the ground distance, the last two parts are the same. The first two parts depend only on one of the operands and can be pre-computed right after the signature extraction. This reduces the necessary computations of the ground distance to $m*n$, which is $\frac{1}{4}$ of the original computations for $m = n$ and even less for $m \neq n$.

- Approximating e^x, which was evaluated in each of $n*m$ iterations. We have $\exp(x) = \lim_{n\to\infty}(1+\frac{x}{n})^n$. We approximated it for $n = 1024$, which leads to 10 ($\log_2 1024$) multiplication operations. This reduces the time computation for a single e^x operation by 31% with average deviation of 0.01%.

With those improvements, the new implementation needs only 7.6% of the time needed by the original implementation, which is a 13× speedup.

4.2.3 Running the measurement

We ran the optimization script on large data-sets, which was very time-consuming. We utilized a 24-core CPU by parallelization of the computations and by supporting multiple instances of the script at once. To avoid duplicate measurements, we used the file system. Each measurement resulted in a file having JSON[4] representation of its configuration as its name. These files contained extracted signatures, so we could quickly update the results by more values of alpha parameter.

To compute MAP we needed a ground truth. Every data-set has a different representation of it, that is why we attached *class name* at the beginning of every signature.

[4]Javascript Object Notation. A lightweight data-interchange format.

4.3 Server architecture

SIR portal is written using ASP.NET MVC3 web framework, which uses the model-view-controller design pattern. Views are templates using the Razor engine, which enables portions of C# code inside HTML. Controllers handle web requests, bind views to models and send the result to the client. All the application logic is inside models.

To distribute URL requests to controllers, we use the default routing strategy. Each URL is in the form of */controller name/action name/parameters*. Controller name is name of the class and action name is the name of the method which gets invoked.

4.4 Static search

4.4.1 Data-set creation

To create our data-set, we import pictures from *Flickr* using *Flickr API*[5] or we query our snapshot database. Both of these methods produce a list of URLs linking to images and some additional meta-data. The import is a batch operation that can be run by the administrator of SIR portal.

After we get those URLs, we need to download each image, process it with all our plug-ins and save information about it to our Static search database. To increase the throughput of the import we implement this process as a pipeline. We can change the number of threads assigned to downloading and processing the images in the configuration file.

Parallelization of this pipeline results in a producer-consumer problem, where threads downloading images are producers and the rest are consumers. To deal with this problem, we made us of .NET framework's concurrent collection, namely the *ConcurrentBag* class[6].

4.4.2 Plug-in architecture

The plug-in architecture is based on System.Addin[7] model of the .NET framework. The plug-ins can be added by uploading a .dll file to the plug-in folder. Each plug-in has to implement *MyAddinView* interface and be annotated by *[System.AddIn.Pipeline.AddInBase]* attribute. If the plug-in has a reference to SIR project and can be loaded at compile-time, it can directly extend the *SiretPlugin* class instead and evade the overhead of using a plug-in system and runtime loading. To reduce that overhead for dynamically added plug-ins, the loading is cached and produces a series of files in the plug-in folder.

[5] A web service of Flickr accessing some of its functionality. http://www.flickr.com/services/api/

[6] http://msdn.microsoft.com/en-us/library/dd381779.aspx

[7] http://msdn.microsoft.com/en-us/library/bb384200.aspx

4.4.3 Implementing plugins

All our similarity search implementations have a common ancestor and only modify its behavior via changes in parameters. The extracted signatures are stored in a pivot table index, which works in memory. If the index is no longer used, it is serialized to the hard drive. Both the common ancestor and the pivot table index are build in a way to allow very easy modifications. Algorithm 4.4.3 shows the only modification that turned the metric space defined by the SQFD to a non-metric and significantly improving performance of the pivot table index.

Algorithm 4.1 Non-metric distance

```
protected override double distance(FeatureSignature FS1,
    FeatureSignature FS2)
{
  var d = base.distance(FS1, FS2);
  return d * d;
}
```

4.4.4 Query processing

All plug-ins communicate with SIR via a thin interface. Search results are passed as a globally unique identifier (GUID). The SIR portal then maps those GUIDs to information stored in a relational database. To abstract the database, we use *Entity Framework*, which maps database entities to C# objects and vice versa.

The plug-ins must be aware of possible concurrency issues. Our pivot-table implementation makes use of *ReaderWriterLockSlim* [8] class to manage accesses to the index. The only longer write-lock occurs when adding a new pivot, which should occur only during a pre-processing phase.

4.4.5 Client-server communication

A lot of client-server communication happens as an AJAX[9] request. We send only small snippets of HTML code and insert them in the current page, thus saving network traffic.

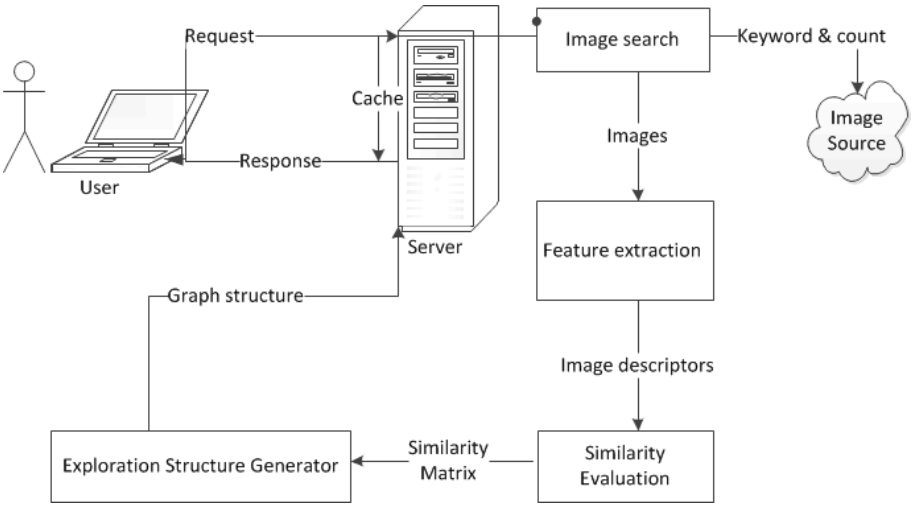


Figure 4.1: Request

4.5 Online re-ranking

4.5.1 Request lifecycle

As we can see in Figure4.1 , the online re-ranking architecture outlines four high-level components. Each component but the first uses output from the previous component and a part of the client's request as its parameters. The request begins with querying an image search engine, in our case Bing. It then continues with downloading each image, extracting its feature signatures using parameters from the request and disposing that image. To eliminate distance computations, we then build a pivot table and compute distances between each pair of images using permutations. Number of pivots, permutation distance or even composition of different permutation distances can be specified in the request. After having all distances, we transform the data and send it to the client script as a JSON object (Algorithm 4.2). To keep our exploration and multi-query algorithm running smoothly on the client machine, we had to enhance the object we send. For simply displaying a set of images as nodes and edges, only a collection of images and a subset of similarities between them was necessary. The exploration feature needs k nearest neighbors for each image and the multi-query algorithm needs also the similarity values between each image and each of its neighbors. This is sent as a $N * K$ matrix of indices to the image collection and same-sized matrix for the similarity values. For the initial zoom-out view we pick the objects at random on the server and send it to the client script as a collection of indices to the image collection.

Each component but the first can utilize parallel environment. By using 6 CPU cores, the average request for a non-cached query spends 68% of the time waiting for the image

[8] *http://msdn.microsoft.com/en-us/library/system.threading.readerwriterlockslim.aspx*

[9] Asynchronous JavaScript and XML. A technology to send data between server and client without the need to reload the page.

Algorithm 4.2 Data structure class

```
public class SimilarityGraph
{
    public Picture[] nodes { get; internal set; }
    public GraphLink[] edges { get; internal set; }
    public GraphLink[] zoomOutEdges { get; internal set; }
    public int[] visibleNodes { get; internal set; }
    public int[][] afterClickNeighbors { get; internal set; }
    public double[][] afterClickDistances { get; internal set; }
}
```

search engine, 27% of the time downloading images and only 5% of the time in all the other parts. By attaching our system to an existing image search engine and thus eliminating all the expensive internet traffic the performance would increase rapidly.

4.5.2 Permutations

We allow the user to set any weighted combination of the three permutation distances we have to form a distance used by the online re-ranking module.

Exact match distance counts the number of positions, at which the two permutations differ.

Kendall Thau distance counts the number of pairwise disagreements between the two permutations. We go trough all possible pairs of values and if their order in the permutations differs, we raise the distance by one. From definition, this distance requires $O(\binom{n}{2})$ time, where n is the length of the permutations. This means that for longer permutations it can be even slower than the original SQFD, which we are trying to approximate to save time.

Position differences distance counts the sum of differences in positions. For each item, we calculate the difference between its position in the two permutations. Also we multiply the difference by a weight according to the lower of the two positions. The highest weight is for the first position and the weights are uniformly decreasing up to the last position. To normalize this distance, we use a simple trick: the distance is divided by the maximum possible distance, which is between the first lexicographical permutation and its reverse permutation.

4.5.3 Force-directed layout

We use the implementation of force-directed layout from Protovis library[12]. We control the layout by changing its nodes, edges and events that occur after user's actions. Once

the page is loaded, the client script written in javascript no longer communicates with the server.

4.5.4 Multi-query algorithm

The multi-query is a query with any positive number of selected images as the input. Let X be the collection of query images, W the collection of their weights and J the desired size of the result. The algorithm works as follows:

Algorithm 4.3 MultiQuery

```
def multiquery(X,W,J)
ranking := {};
for image in X
  for neighbor in KNN(X)
    ranking[neighbor]+= similarity(image,neighbor) * W[image]
Optionally exclude query images from the ranking set
Return  J images with the highest value from ranking.
```

Notice that by using a negative weight we can even penalize all neighbors of a query image with the negative weight. If we have precomputed $KNN(X)$ with similarity values for each object in our data-set and K is a small constant, the algorithm runs very fast. Let N be $|ranking|$ which is upper bounded by $K * |X|$. Ranking of the images is done in $O(K * |X|)$ time and taking the best-ranked images can be done in $O(N + J * logN)$ time using a max-heap. However, for smaller values in practise, it is faster to use $O(N \log N)$ quicksort instead of the max-heap.

Conclusion

We have created a web portal for content-based image retrieval called Smart Image Retrieval. We have shown, that synergistic modeling can help us find similarity spaces with both good precision and good indexability and utilized the results in the static part of SIR. In the other part, we designed an image meta-search engine that allows content-based exploration of the results. The key features of the online re-ranking are a very fast feature extraction and the presentation of results based on particle physics model and data-set exploration.

Future work

In the future, we plan to extend the datasets SIR can explore and include hierarchical exploration based on a tree index as well. We also plan to use online reranking module for other data, such as custom image galeries or images of products imported with a XML file. In addition to that, we would like to improve the quality of feature extraction by using other image features such as texture and train that feature extraction on real-search data using our snapshot database.

Attachments

The following documents can be found on the attached CD-ROM:

- PDF version of this thesis
- Source code to synergistic modeling script
- Measured results as .csv and .xlsx files
- Source code to SIR web portal
- SIR web portal's binary files and a sample database

Nomenclature

AJAX	Asynchronous JavaScript and XML. A technology to send data between server and client without the need to reload the page.
ALOI	Amsterdam Library of Object Images
API	Application interface.
CBIR	Content based image retrieval
GUID	Globally unique identifier
HTML	HyperText Markup Language. A language for creating web sites.
iDim	Intrinsic dimensionality, a single value indicating quality of distance distribution for indexability.
JSON	Javascript Object Notation. A lightweight data-interchange format.
KNN	The nearest neighbor query returns K closest objects to the query object.
MAP	Mean average precision
SIR	Smart Image Retrieval. Name of the web portal proposed in this thesis.
SQFD	Signature Quadratic Form Distance
URL	Uniform Resource Locator. A URL is the address of a specific web site or file on the Internet.

Bibliography

[1] P. Zezula, G. Amato, V. Dohnal, and M. Batko. Similarity Search: The Metric Space Approach. Springer-Verlag New York, Inc., 2005.

[2] M. Kruliš, J. Lokoč, Ch. Beecks, T. Skopal, T. Seidl. Processing Signature Quadratic Form Distance on Many-Core GPU Architecture, CIKM 2011.

[3] Margulis, Dan. Photoshop Lab Color: The Canyon Conundrum and Other Adventures in the Most Powerful Colorspace, 2006.

[4] Brin, S. and Page, L. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In: Seventh International World-Wide Web Conference (WWW 1998), April 14-18, 1998, Brisbane, Australia.

[5] L.-J. Li and L. Fei-Fei. Optimol: Automatic online picture collection via incremental model learning. Int. J. Comput. Vision, 88(2):147-168, June 2010.

[6] Fergus, R. , Fei-Fei L. , Perona, P. and Zisserman, A. Learning Object Categories from Google's Image Search. Proc. of the 10th Inter. Conf. on Computer Vision, ICCV 2005.

[7] Fergus, R. , Perona, P. and Zisserman, A. A Visual Category Filter for Google Images. Proc. of the 8th European Conf. on Computer Vision, ECCV 2004.

[8] Lowe, David G. Object recognition from local scale-invariant features, Proceedings of the International Conference on Computer Vision. 2. pp. 1150–1157, ICCV 1999.

[9] Brin, S.. Near neighbor search in large metric spaces. In: Proc. 21st Conference on Very Large Databases (VLDB95). pp. 574–584,1995.

[10] Susan Sim. Automatic Graph Drawing Algorithms, Information Visualization, 1996

[11] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. Software: Practice and Experience, 21(11):1129–1164, 1991.

[12] M. Bostock and J. Heer. Protovis: A Graphical Toolkit for Visualization, http://mbostock.github.com/protovis/protovis.pdf, 2010.

[13] Skopal T., Pokorný J., Snášel V.: PM-tree: Pivoting Metric Tree for Similarity Search in Multimedia Databases, in ADBIS, Computer and Automation Research Institute Hungarian Academy of Science, ISBN: 963 311 358 X, pp. 99-114, 2004

[14] Ciaccia, Paolo; Patella, Marco; Zezula, Pavel. M-tree An Efficient Access Method for Similarity Search in Metric Spaces. Proceedings of the 23rd VLDB Conference Athens, Greece, 1997.

[15] C. Beecks, J. Lokoč, T. Seidl, and T. Skopal. Indexing the signature quadratic form distance for efficient content-based multimedia retrieval, ICMR 2011.

[16] J. Lokoč, M. Hetland, T. Skopal, and C. Beecks. Ptolemaic indexing of the signature quadratic form distance, SISAP 2011.

[17] T. Skopal. Unied framework for fast exact and approximate search in dissimilarity spaces. ACM Trans. Database Syst., 32(4):46, 2007.

[18] T. Skopal On Fast Non-Metric Similarity Search by Metric Access Methods, EDBT 2006, Munich, Germany, LNCS 3896, Springer.

[19] C. D. Manning, P. Raghavan, and H. Schutze. Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA, 2008.

[20] Tomáš Grošup, Jakub Lokoč, Tomáš Skopal. Image Exploration Using Online Feature Extraction and Reranking, ICMR 2012.

[21] Tomáš Grošup, Jakub Lokoč, Tomáš Skopal. SIR: The Smart Image Retrieval Engine, accepted to SISAP 2012.

[22] Jakub Lokoč. Tree-based Indexing Methods for Similarity Search in Metric and Non-metric Spaces, Doctoral Thesis, Charles University in Prague, 2010.

[23] Moravec, P. Testing Dimension Reduction Methods for Text Retrieval. In Proceedings of Dateso'05 Workshop.

[24] J. M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders, The Amsterdam library of object images, Int. J. Comput. Vision, 61(1), 103-112, January, 2005.